어려운 프로그래밍, 불가능한 프로그래밍, 그에 따라 변화하는 프로그래밍 트렌드에 대하여

박선호

September 6, 2022

Abstract

수학의 완전성, 일관성, 판정가능성은 여러 수학자들에 의해 부정되었고, 수학과 밀접한 관련이 있는 프로그래밍 역시 이것에 영향을 받아 한계가 존재한다. 또한 같은 이유로 소프트웨어의 정확성(correctness)을 증명하는 것 역시 어렵다. 이러한 어려움이 있어도, 우리는 문제의 조건을 완화하거나 근사해를 찾는다는 방식으로 문제를 최대한 해결해 나가야 하며, 실제로 이러한 방향으로 프로그래밍의 트렌드가 변화하므로 발빠르게 맞춰 나가야 한다.

여러분들은 컴퓨터에 대해 잘 알지 못했던 시절, 컴퓨터는 무슨 일이든지 쉽게 해내는 마술 상자라는 생각을 한 적이 있는가? 어릴 적부터 컴퓨터를 좋아하여 이것저것 공부를 했던 필자는 줄곧 그렇게 생각해 왔다. 그도 그럴게, 전 세계의 대단한 엔지니어들이 몇십년간 엄청난 기술 발전을 이루어냈기 때문이다. 과거에 수학의 완전성, 일관성, 판정가능성을 주장한 수학자 힐베르트(David Hilbert, 1862 - 1943)는 묘비에 "우리는 반드시 알 것이며, 알게 될 것이다"(We must know, we shall know)[1] 라는 말을 남겼다. 이와 마찬가지로 프로그래밍을 잘 모르는 일반 사람들이 컴퓨터와 프로그램에게 바라는 요구사항으로는, 위의 말을 패러디하자면 "우리는 반드시 해결할 것이며, 해결하게 될 것이다"(We must solve, we shall solve) 일 것이다. 즉, 어떠한 어려운 문제가 있어도 프로그래머들이 소프트웨어를 만들 수 있어야 하며, 만들어진 프로그램을 돌리면 척척 문제없이 해결할 수 있어야 하는데, 여러분들은 이것이 정말로 가능하다고 생각하는가?

결론부터 얘기하자면, 현실은 그렇게 호락호락하지 않다. 즉, 소프트웨어를 설계해도 영원히 해결할 수 없는 문제가 있다는 말이다. 단순히 합리적인 시간 안에 풀 방법이 알려져 있지 않은 문제만 있는 것이 아니다. 세상에 어떠한 문제들은 합리적인 시간 안에 풀 수 있을지 아닐지 증명에 성공한 사람이 없어서 난제로 남아 있기도 하고, 심지어 어떤 문제는 완벽하게 풀 수 없다는 사실이 증명된 문제들도 있다. 쉽게 생각하면 판정불가능한(undecidable) 문제들이 그 예시가 된다. 정지 문제(Halting problem)가 첫 번째로 증명된 판정불가능한 문제이다. 정지 문제란, 어떠한 프로그램과 초기 입력이 주어졌을 때 프로그램이 유한 시간 안에 동작을 정지하는지 아닌지 판정하는 문제인데, 1936년 앨런 튜링(Alan M. Turing, 1912 - 1954)에 의해 판정불가능한 문제임이 증명되었다. 보다 현실에 가까운 예시를 들자면, '완전한 완전성(completeness)와 안전성(soundness)을 가지는 타입 체계(type system)를 만들 수 있는가?'라는 문제가 있다. 타입 검증(type checking)은 프로그램을 실행하기 전 컴파일 단계에서 프로그램의 에러를 탐지하기 위해 도입된 기술로, 여기서 완전성은 타입 검증 단계에서 거부된 프로그램은 모두 실제로 에러를 포함하고 있다는 성질을 뜻하고 안전성은 타입 체계가 모든 에러를 포함한 프로그램을 거부한다는 성질을 뜻한다. 이 두 가지 성질을 모두 만족하기 위해서는 모든 프로그램에 대해서 실행하기 전에 에러가 발생할지 여부를 정확하게 판단해야 하는데 이 역시 정지 문제와 마찬가지로 라이스의 정리 (Rice's theorem)에 의해 판정불가능한 문제임이 증명되어 있다.

이렇게 소프트웨어로 풀 수 없는 문제가 존재하는 근본적인 이유로 나는 수학의 한계 때문이라고 생각한다. 형식주의자였던 힐베르트는 수학의 힘을 믿고 수학의 완전성, 일관성, 판정가능성을 주장했다. 여기서 완전성이란 모든 참인 명제는 증명가능하다는 성질이고, 일관성은 모순이 없다는 성질이며, 판정가능성은 어떠한 명제가 공리를 따르는지 여부를 결정할 수 있는 알고리즘이 존재한다는 성질이다. 하지만 이후에 수학자 괴델(Kurt Friedrich Gödel, 1906 - 1978)의 불완전성 정리(incompleteness theorem)에 의해 완전성과 일관성이 부정되었고, 판정가능성은 정지 문제에 의해 처음으로 부정되었다. 이러한 상황에서 소프트웨어의 한계가 수학의 한계와 밀접한 관련이 있다고 생각한 이유는, 프로그램의 동작은 프로그래밍 언어의 의미 구조(semantics)로부터 추론

[1] "David Hilbert", Wikipedia, `https://en.wikipedia.org/wiki/David_Hilbert#Death`

규칙(inference rule) 등과 같은 논리적인 방식으로 엄밀하게 이해할 수 있는데 이러한 논리적인 방식이 곧 수학적 사고방식이기 때문이다. 바로 윗 문단에서 서술했듯이 정지 문제나 완전성과 안전성을 확보한 타입 체계의 불가능성 역시 수학의 판정불가능한 성질이 허용한 결과임을 확인할 수 있다.

소프트웨어의 정확성(correctness)을 증명하는 것 또한 수학의 한계 때문에 어려운 과정이라고 생각한다. 프로그램의 정확성은 항공기의 핵심 프로그램, 로켓에 사용되는 프로그램 등 안정성이 꼭 보장되어야 하는 경우에 반드시 증명하여야 하는 문제인데, 이러한 프로그램의 정확성을 증명하기 위해서는 주어진 프로그램의 사양(specification)을 구상한 다음, 프로그램의 가능한 모든 실행 흐름(control flow)에서 주어진 사양을 만족한다는 사실을 프로그램을 실행하지 않고 논리만을 이용하여 증명하여야 한다. 여기서, 프로그램의 사양이 주어졌을 때 증명을 알고리즘적으로 쉽게 해결할 수 있다면 프로그램의 정확성을 증명하는 과정 또한 굉장히 쉽다는 증거가 되겠지만, 프로그램이 주어진 사양을 만족하는지 판정하는 문제보다 쉬운 문제인 프로그램이 에러를 일으키는지 판정하는 문제가 이미 판정불가능한 문제임이 증명되어 있으므로 자동으로 증명을 마쳐주는 알고리즘 역시 존재하지 않음을 알 수 있다. 또한 어쩌면 수학의 완전성이 괴델의 불완전성 정리에 의해 부정되었듯, 어떠한 프로그램에 대해서는 사양을 증명하지 못하는 경우가 있을지도 모른다. 그러므로 소프트웨어의 정확성을 증명하는 것은 매우 어려운 과정이라고 볼 수 있다.

이렇듯 소프트웨어의 설계와 정확성 증명 과정 모두 어렵고 때로는 불가능한 경우도 있는데, 그렇다고 하더라도 우리는 풀어야 하는 문제가 있으면 엔지니어적인 방식으로 문제의 조건을 완화해서 풀거나, 아니면 근사해를 찾는 방식으로 끝까지 해결하도록 노력해야 한다고 생각한다. 사실 이러한 방식은 전 세계적으로 최근에 굉장히 활발하게 사용하고 있다. 그 예시가 바로 인공지능이다. 바둑 경기에서 이세돌 기사를 이긴 알파고(AlphaGo)라는 인공지능 역시 바둑 경기에서 최선의 수를 찾는 알고리즘이 존재하더라도 현실적인 시간 안에 계산하기 힘드므로 학습 과정을 거친 인공지능을 이용하여 근사해를 찾는 방식으로 바둑 경기를 진행하여 승리를 거머쥐었다. 마찬가지로 프로그램의 모든 에러를 컴파일 시간 동안 타입 체계를 이용하여 알고리즘적으로 모두 탐지하는 것이 불가능하므로 프로그램의 정적 분석이라는 분야에서는 여러 휴리스틱 알고리즘이 고안되어 왔고 최근에는 인공지능을 이용하려는 시도가 많이 이루어지고 있다. 그 외에도 이미지 분석, 번역, 음성 인식 등 알고리즘을 설계하기 힘들어 보이는 문제를 인공지능을 이용하여 근사적으로 문제를 해결해 왔고, 실제로 이러한 접근법이 현대인의 삶을 크게 윤택하게 만들어 주었음은 부정할 수 없다. 그러므로 앞으로는 이러한 근사적으로 문제를 해결하는 데 초점이 맞춰질 것임이 분명하고, 이러한 트렌드에 맞춰 나아가는 것이 중요하다는 점을 강조하고 싶다.